Title

Memory-Based High-Level Synthesis Optimizations Security Exploration on the Power Side-Channel

Journal

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 39(10)

ISSN

0278-0070

Authors

Zhang, L
Mu, D
Hu, W
et al.

Publication Date

2020-10-01

DOI

10.1109/TCAD.2019.2950380

Peer reviewed

Memory-Based High-Level Synthesis Optimizations Security Exploration on the Power Side-Channel

Lu Zhang, Dejun Mu, Wei Hu, Yu Tai, Jeremy Blackstone, and Ryan Kastner

***Abstract*—High-level synthesis (HLS) allows hardware designers to think algorithmically and not worry about low-level, cycle-by-cycle details. This provides the ability to quickly explore the architectural design space and trade-offs between resource utilization and performance. Unfortunately, security evaluation is not a standard part of the HLS design flow. In this work, we aim to understand the effects of memory-based HLS optimizations on power side-channel leakage. We use Xilinx Vivado HLS to develop different cryptographic cores, implement them on a Spartan-6 FPGA, and collect power traces. We evaluate the designs with respect to resource utilization, performance, and information leakage through power consumption. And we have two important observations and contributions. First, the choice of resource optimization directive results in different levels of side-channel vulnerabilities. Second, the partitioning optimization directive can greatly compromise the hardware cryptographic system through power side-channel leakage due to the deployment of memory control logic. We describe an evaluation procedure for power side-channel leakage and use it to make best-effort recommendations about how to design more secure architectures in the cryptographic domain.**

***Index Terms*—Hardware security, high-level synthesis, power side-channel evaluation, design space exploration.**

I. Introduction

HIGH-level synthesis (HLS) allows a designer to quickly restructure their code using high-level behavioral descriptions or instruct the tool to perform automatic architectural optimizations such as data partitioning, pipelining, and unrolling [1]. This enables her to rapidly generate different architectures and explore a large design space [2]. This, along with the availability of mature commercial HLS tools, has lead to wider adoption of HLS in the hardware design process. Using these tools, it can emancipate programmers from extensive hand-coding in RTL and manual tuning. To improve the quality of the HLS design flow, many possible combinations of optimization strategies are introduced and evaluated, resulting in many insightful guidelines for the HLS programming community.

Manuscript received February 16, 2019; revised April 28, and July 29, 2019; accepted October 8, 2019. This work is partly supported by NSF of China, National Cryptography Fund of China, Innovation Fund of Shenzhen Research Committee, the Shaanxi Provincial Key R&D Program, Technology Project of State Grid Corporation of China under grant 61672433, MMJJ20170210, 201703063000517, 2018KW-005, 522722180007. This project is partly supported under NSF grants CNS-1563767, CNS-1527631, and CNS-1718586.

L. Zhang, D. Mu, W. Hu, and Y. Tai are with the School of Cyberspace, Northwestern Polytechnical University, Xian, Shaanxi 710072, China (e-mail: willvsnick@gmail.com; mudejun@nwpu.edu.cn; weihu@nwpu.edu.cn; taiyu@mail.nwpu.edu.cn).

J. Blackstone and R. Kastner are with the Department of Computer Science and Engineering, University of California, San Diego, CA 92093 USA (e-mail: jblackst@ucsd.edu, kastner@ucsd.edu).

Cryptographic algorithms are commonly implemented in hardware to improve throughput and power consumption [3]. This has naturally prompted studies on how different cryptographic algorithms and architectures compare with respect to performance, power consumption, and resource usage [4]. Many cryptographic cores naturally map into HLS languages, making it an attractive approach for designing cryptographic hardware. While it is easy to measure the performance, power, and resource usage, there is not a standard, built-in way to determine the security of a particular design [5]. It is thus important to understand how these HLS optimizations effect the design's security alongside the traditional power, performance, and resource usage metrics. This is especially important in the cryptographic domain where there are significant security concerns regarding to side-channel leakage [6].

Power side-channels are one of the most exploited security vulnerabilities for cryptographic hardware. This has been studied for decades, and it is well-known that an attacker can extract confidential information using a (often very simple) statistical analysis of the computation's power consumption [7]. As a consequence, there have been a large number of defenses against these power side-channel attacks including masking and hiding [8], [9]. As these defenses get implemented, the attacks become more sophisticated. This presents a game of "cat and mouse" where designers attempt to mitigate the vulnerabilities with more sophisticated defenses at the same time that attackers perform more complex attacks. HLS technique allows one to quickly generate different architectures and employ various defenses. However, this requires an understanding of the trade-offs when developing cryptographic systems using HLS tools. This is the question that we aim to understand: how do we effectively leverage HLS to create fast, small, and secure cryptographic hardware module?

State-of-the-art HLS tools such as Vivado HLS [10] or LegUp [11] deliver a rich set of local synthesis directives that can optimize your design by providing many alternative choices for design space exploration (DSE) [12]. This work aims to better understand the implications of these local HLS optimizations on the power side-channel. However, it is much harder to distinguish the side-channel effects of each local optimization directive individually due to the flexible nature of HLS and the complexity of the HLS-generated architectures. Moreover, with the exponential growth of design space, it is always time-consuming and costly to do security evaluation for HLS DSE. To address the limitations, we build a standard framework to evaluate the side-channel effects of memory-based HLS optimizations. As a result, the side-channel effects of each HLS optimization directive become traceable and

controllable with respect to side-channel analysis (SCA).

How power side-channel vulnerabilities are formed during HLS procedure is a key question to answer for the security analysis. Essentially, the reason why most SCA attacks mainly focus on the non-linear module in crypto-algorithms is due to the facts that glitches happening inside the implementation of non-linear substitution contribute a large fraction of dynamic power leakage. Meanwhile the substitution operation correlates more with the implementation of internal memories during HLS transformation. Thus, the implementation of S-boxes provides a natural starting point for understanding the effects of HLS optimization, and we mainly focus on S-box optimization. Yet, how the HLS optimization directives would affect the power vulnerabilities as well as the overall performance of a specific design is still unknown.

To address this issue, our work represents a first step to place the questions on an empirical, quantifiable basis. The first issue we tackle is whether different HLS optimizations change the power side-channel leakage. This answer is not surprising – yes, they do. The second one is to understand why the HLS optimization affects the security of the cryptographic design. The third one is comparing HLS designs with well-known RTL-based cryptographic designs and determining any differences between these architectures generated using design entry at different levels of abstraction. Our best-effort guidance can substantially decrease the risks to generate an architecturally insecure design, providing more security insights for designers. This paper is an extended version of the work published in [13], the main contributions are as follows:

- Providing a framework to evaluate power side-channel leakage as a security metric when performing HLS;
- Performing an in-depth security analysis on HLS partitioning technology in cryptographic domain by exploring the side-channel effects of different combinations of HLS optimization directives using real-world power traces;
- Presenting the first experimental results that qualitatively and quantitatively evaluate the consequences of memory-based HLS optimizations;
- Demonstrating the best design trade-offs among the HLS-generated benchmarks, comparing between HLS and RTL benchmarks, and finally revealing more insights of HLS optimizations on power side-channels.

The reminder of the paper is organized as follows. Section II presents the preliminaries of behavioral synthesis and block ciphers with respect to SCA security. Section III demonstrates the basic criteria for SCA evaluation and the evaluation metrics. Section IV shows the security evaluation workflow and the reference architecture for HLS DSE. Section V shows the general process for pragma-based HLS DSE and the description of benchmarks. In Section VI, we analyze and compare the experimental results. At last, we briefly review related work in Section VII, and conclude in Section VIII.

II. Preliminaries

This section provides the background. We first describe the basic transformations of behavioral synthesis and discuss the corresponding security concerns. Then, we introduce the Substitution-Permutation networks widely used in block ciphers and its correlation with SCA attacks. Finally, we discuss the limitations and solutions for the HLS DSE.

A. Behavioral Synthesis and Potential Security Challenges

Behavior synthesis seeks to automatically translate high-level languages to register transfer level (RTL) expressions so as to reduce the design effort. The main concerns, when converting behavioral descriptions in HLS to micro-architectures in RTL, are the transformations of arrays, loops, and functions. Fig. 1 represents a motivational example, which demonstrates the main transformations of HLS in detail as well as the potential security challenges for each type of transformation. In general, 1) Arrays in HLS description are always expressed as memories in RTL description, and then implemented using Block/LUT RAMs as storage instances. For example, the memory-based optimization can guide the HLS tools to specify the types of implementation (memories or registers) as well as the number of ports (single-port or dual-port) for RAMs. 2) Loops in HLS have great effects on how to create a finite state machine (FSM), thus generating the corresponding control logic, and the loops also affect the design hierarchy. For instance, the loop-based optimization can decide how the loops are unrolled or pipelined, which mainly affects the timing constraints and the throughput. 3) Functions are the main body of behavioral description as it can define the main hierarchy of design as well as the I/O ports or communication protocols. For example, the function-based optimization can specify whether two functions of a design should be inlined.

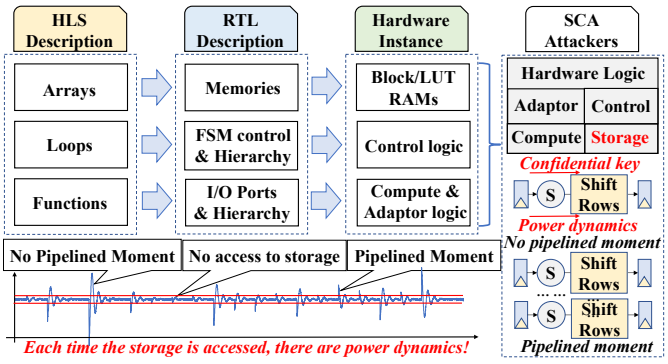


Fig. 1. The HLS translation workflow from the behavioral description to the hardware instances and also the DPA implications of the HLS design flow.

The inherent relation between power side-channel leakage and HLS workflow can be explained as follows. In terms of the cryptographic security, the corresponding local optimization directives can be divided into two categories: first, the memory-based optimization directives (e.g., resource or partitioning) can directly affect how to construct a memory or decide the layout of multi-bank memory architectures. Consequently, by carefully exploring how to synthesize or implement this sub-architecture, the implication of side-channel leakage in terms of HLS can be revealed. Second, the loop- or function-based optimization directives (e.g., loop pipeline or function inline) can affect the side-channel vulnerabilities by changing the hierarchy of a design or by affecting the

FSM control. More precisely, by changing the *substitution* highlighted in red in Fig. 2, the memory-based optimization directives could directly affect the formation of side-channel leakage. For example, as shown in Fig. 2, the leakage peaks only appear in the first/third clock cycle, where the *substitution* highlighted in red starts to operate. Note that, power dynamics and confidential information are two essential ingredients in the formation of side-channel leakage. Namely, only if the power dynamics correlate with the confidential information (e.g., the case of substitution), it is considered as side-channel leakage. Otherwise, it is considered as the noises.

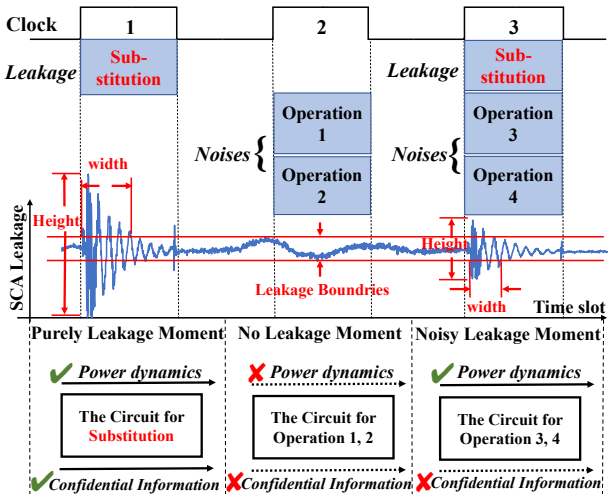


Fig. 2. The SCA leakage comparison between purely leakage moment, no leakage moment and noisy leakage moment.

A major challenge of security evaluation in behavioral synthesis is derived from the architectural complexity and flexibility of HLS. Though some modifications in HLS description are so tiny, it can completely change the corresponding expressions in RTL level, finally leading to many architectural changes along with the security risks in hardware implementation. Moreover, it is especially difficult to judge whether the side-channel leakage changes are caused by a single factor or criterion. For example, inserting a scheduling-based optimization directive (e.g., pipeline or unroll) to the HLS description may dramatically affect the hierarchy of design as well as the FSM control. In this case, it is extremely difficult to distinguish the side-channel effects of each HLS optimization directive separately as the architectural changes are made everywhere also depending on the implementation rules provided by HLS tools. For the memory-based optimization, it might not incur too many changes in the hierarchy or the FSM control. Yet, the appearing challenge is lack of a golden noise-free reference design, facilitating a fair and efficient security comparison among these micro-architectures.

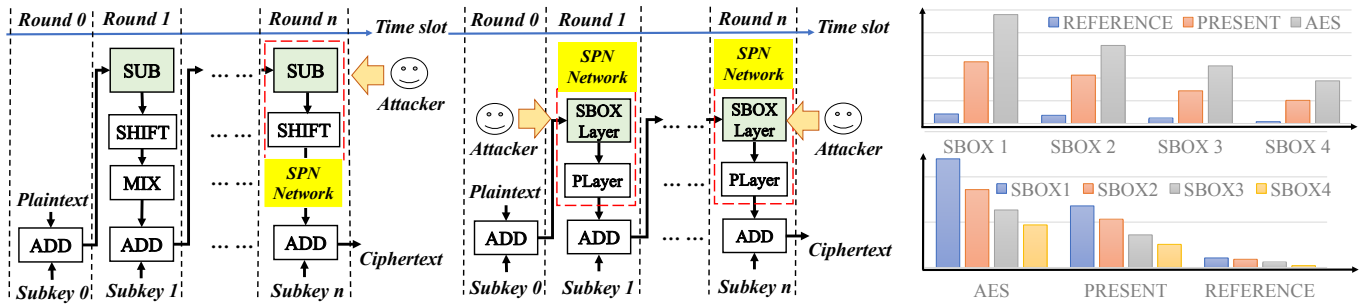
B. Block Ciphers and Problem Formulation

Block cipher is defined as a deterministic algorithm performing on fixed-length groups of bits, called a block, with an unvarying transformation that is specified by a symmetric key. By combining simple operations such as substitutions and permutations, the security level of such algorithms can be effectively enhanced. Therefore, the Substitution-Permutation Networks (SPN) operate as both important and widely adopted elementary structure in the design of modern block ciphers, aiming at satisfying the Shannon's "Confusion" and "Diffusion" properties. Generally, the "Confusion" is performed by a layer of S-boxes to achieve the non-linear transformation. The "Diffusion" is performed through the linear permutation called P-box. Meanwhile, several alternating "rounds" or "layers" of SPN structures are performed in order to generate the final ciphertext. The best example for such SPN structure is Rijndael, the cipher which has been standardized to become the AES. Indeed, there exist various versions of block ciphers that are based on the SPN-based structure (e.g., PRESENT).

Depending on the application scenario, there might exist some architectural differences among those block ciphers. However, given the nature of SCA, most side-channel attackers only choose the "Confusion" operation of one chosen time-slot (e.g, a part of the non-linear transformation layer) as a target, rather than the entire cipher. The typical SCA scenario assumes an attacker already knows the cipher plaintexts or ciphertexts, the "Confusion" operation is generally indexed with values depending from both plaintexts (or ciphertexts) and a portion of the secret key. Subsequently, by comparing the power prediction taken by the portion of the secret key with the measured power consumption, the actual value of the key portion can be revealed through the use of a statistical tool. As the "Confusion" causes more power consumption due to the complex implementation of S-boxes, it represents an easy information leakage spot for attackers. While "Diffusion" is implemented only using simple wiring, it has little effects on leakage dynamics. Fig. 3 shows a motivational example.

Fig. 3 (c) represents the MTD (Measurement-To-Disclosure) comparison between two well-known block cipher Rijndael AES and PRESENT, as shown in Fig. 3 (a) (b), and our proposed reference design (REFERENCE), as shown in Fig. 5 (a). Two important observations can be obtained from this example. 1) Observation 1: For different ciphers, attackers firstly have to decide the suitable time-slot to launch the attack. For example, the last round of Rijndael AES is always chosen as the targeted time-slot, since it is the only round for AES without the operation of "MixColumns", in which the leakage arising from the operation of "SubBytes" can achieve the maximization. PRESENT has more intermediate states to target due to its rather simple implementation. 2) Observation 2: The stand or fall of a design is not specific to different types of ciphers but rather to the fact that how to implement the S-boxes using memory primitives inside, including not only the specific layout of memories but also the unique structure of each memory. We stress that our analysis is not limited to cryptographic ciphers such as AES or PRESENT, but are also relevant for any other block ciphers in which the similar SPN-based structure is adopted.

This paper makes use of those observations to facilitate the HLS DSE. Based on observation 1, we obtain the experience that the first challenge for attackers is to locate the suitable time-slot (e.g., a small chunk of the power trace), when the FSM starts to call the S-boxes. As a result, the problem for attackers is converted to a problem of deciding when the



(a) Block Cypher AES. n ∈ { 10, 12, 14} (b) Block Cypher PRESENT. n ∈ { 26, 31} (c) MTD Comparison between AES, PRESENT and REFERENCE

Fig. 3. The function diagrams of the block ciphers (a) AES (the key expansion omitted) and (b) PRESENT (the key expansion omitted). (c) the MTD metric comparison between block cipher AES, PRESENT and the proposed noise-free reference architecture (REFERENCE).

S-boxes start to operate. From observation 2, we can know that the different constructions of S-boxes do deliver different amount of side-channel vulnerabilities. However, evaluating the side-channel effects of memory-based HLS optimizations on block ciphers is always time-consuming and error-prone. Because it is inherently hard to evaluate the effect of a single optimization directive without considering the effects of other architectural changes that simultaneously happen to the entire design. Therefore, we propose an alternative approach, which facilitates the security evaluation for DSE by purely focusing on the construction of substitution subsystem that is generated by HLS tools. However, one of the main challenges in pragma-based HLS DSE resides in how to provide a fair evaluation for different optimization directives. To address these limitations, there exists an urgency to design a golden noise-free architecture, which allows us to perform a pragma-based DSE and separately evaluating the effects of each memory-based HLS optimization directive on the power side-channel, while regardless of the timing concerns caused by other tasks.

C. Overview of Memory-based Optimization in HLS Tools

There exist various commercial or academic HLS tools [14] in literature, which construct the memory sub-system in their specific way. DWARV [15] is an academic HLS tool that provides many *performance-optimized* options. It has flexible strategies to optimize the memory architectures. For example, it allows to allocate the variables or arrays to external memory (e.g., local on chip buffers) or internal memories (e.g., BRAMs) by configuring the corresponding *pragmas*. LEGUP [11] is an academic HLS tool that allows the use of optimization in the form of *Tcl*. For example, it allows the use of memory partitioning, which is similar to the ones of Vivado HLS. BAMBU [16] is an academic HLS tool that tries to build a hierarchical data-path to a dual-port BRAM. This memory infrastructure facilitates the connection to external devices or to the bus linking the off-chip memories. Vivado HLS [10] is a maturely commercial HLS tool that provides many parameter-based mapping optimization to memories. Among the HLS tools, the commercial Vivado HLS is a more powerful one in that it supports more features and being more robust than the academic tools. Therefore, we take the *performance-optimized* optimization of Vivado HLS as an exemplar.

III. Power Side-Channel Leakage Detection and Metrics Description

In this section, we first introduce the related work in HLS for security that is emerging recently. Then we introduce the SCA evaluation methodologies and the metrics.

A. Emerging Interests of Security with HLS

Modern chip design methods are lack of the necessary investment in scalable security mechanisms, increasing the risks to be hacked (e.g., side-channel attacks). Hence, the program of Automatic Implementation of Secure Silicon (AISS) was proposed by DARPA in order to facilitate the automatic exploration of economics versus security trade-offs. The novelty of this approach is due to the facts that design complexity and cost of integration make it nearly impossible to address newly security issues after finishing the entire design flow. Therefore, AISS aims to provide an approach being beneficial for rapidly evaluating architectural alternatives that best address the required design and security metrics, as well as building cost models to optimize the economics versus security tradeoff. The methodology for security evaluation of HLS proposed in this paper provides a means of evaluating side-channel effects of architectural changes due to HLS optimization and it is beneficial for providing more solutions to solve the problems existed in AISS, simultaneously bringing greater automation to the security design process. As a result, the burden of security inclusion can be largely decreased.

B. Examining the Side-Channel Leakage

Side-Channel leakage evaluation aims to categorize an implementation based upon its vulnerability to attack. For first-order leakage detection, the non-specific t-test is considered as the most common assessment for early-stage assessment [17]. Intuitively, one virtue of non-specific t-test is its independence from any power model and intermediate value. Hereby, it is adopted as a standard method to investigate the existed first-order power side-channel leakage. The basic idea of non-specific t-test is to check if two datasets have identical mean and variance. In such a test, two datasets (D_1 and D_2) are available. Data in D_1 is collected by feeding identical plaintext to the encryption module for m times. While Data in D_2 is measured by sequentially feeding various plaintexts n times.

Yet, the key for D_1 and D_2 is a constant value. A Welch's t-test [18] is performed by computing the equation (1).

$$\hat{t}(d_1, d_2) = \frac{\hat{\mu}_{d_1} - \hat{\mu}_{d_2}}{\sqrt{\frac{\hat{\sigma}^2_{d_1}}{m} + \frac{\hat{\sigma}^2_{d_2}}{n}}} \quad (1)$$

where $d_1 \in D_1$ and $d_2 \in D_2$, and $\hat{\mu}$ and $\hat{\sigma}$ represent the sample mean and sample variance, respectively. It is noteworthy that, from the prospective of practical experience, the null hypothesis of non-specific t-test could be rejected with sufficient evidence only when the result value $|t| > 4.5$. Therefore, the existence of a leakage could be detected.

C. Quantifying Side-Channel Leakage as a Metric

Side-channel attack was first proposed by Kocher in 1999 by exploiting Difference-of-Means test as distinguisher [19]. Among all non-profiled first-order side-channel methods, Correlation Power Analysis (CPA) is considered as the most efficient and optimal one against first-order side-channel leakage [20]. To quantify the security of a specific design, we apply the number of Measurements-to-Disclosure (MTD) as the metric for side-channel security evaluation. In such an attack, randomly generated plaintexts are chosen and fed to the encryption module continuously. While the encryption key is constant for all the measurements. During encryption, power traces are recorded and then correlate to power consumption predicted by the power model. Such comparison can be fairly performed by means of Pearson's Correlation Coefficient for each key hypothesis $\hat{\mathrm{k}}$, using equation (2).

$$\hat{\rho}(r, h_{\hat{k}}) = \frac{cov(r, h_{\hat{k}})}{\hat{\sigma}(r) \cdot \hat{\sigma}(h_{\hat{k}})} \quad (2)$$

where r and $h_{\hat{k}}$ denote the real recorded measurement and hypothetical power consumption, respectively. While the covariance and standard deviation are denoted as cov and $\hat{\sigma}$, respectively. In a successful case, the key hypothesis $\hat{k}_c$ corresponding to the correct guess can lead to a significant $\hat{\rho}_c$ ($\hat{\rho}_c \in (-1, 1)$) value by a large amount than a wrong guess. During the MTD ranking process in HLS DSE, a higher MTD value indicates that the benchmark is more secure in terms of the SCA security. Note also that, if the CPA attack on a specific architecture can not lead to any key recoveries, the MTD value of the design is marked as 100,000+ or SD.

IV. Proposed Framework for Power Side-Channel Evaluation in High-Level Synthesis

In this section, we introduce the proposed framework for side-channel evaluation in HLS and demonstrate the implementation details of the reference architecture.

A. General Side-Channel Evaluation Workflow

Our overall workflow is shown in Fig. 4. The goal is to create different architectures, collect consistent power traces, and evaluate each architecture's resilience to power SCAs. The upper row of Fig. 4 shows the workflow of building different memory structures depending on the *#pragma resource* configuration in HLS-readable source codes. First, we create a C code reference design for "SubBytes" module. Second, these C codes are modified manually to create HLS-readable codes. Third, Vivado HLS generates RTL from the input HLS codes. The lower row of Fig. 4 presents the workflow of deploying the partitioning strategies that results in different memory layout. First, a *#pragma array_partition* directive is added to the source code, where the *#pragma resource* directive is located. Second, by changing the tunable options both in *resource* and *array_partition* directive, we can generate the partitioning benchmarks that belong to the corresponding *pragma-resource* series. After this point, a similar procedure is performed for all the benchmarks. Afterward, we extract the corresponding memory sub-system from HLS-converted RTL and implant it into the reference architecture in Fig. 5 (a). Then these benchmarks are synthesized and implemented using ISE Design Suite v14.7 in turn to obtain resource utilization and performance information, respectively. Finally, we run these benchmarks on the side-channel evaluation *SAKURA-G* board and collect traces for further security evaluation. For security evaluation, we start our work by performing the non-specific t-testing, as described in Section III-B. Simultaneously, we perform a first-order CPA attack to quantify the power leakage as a security metric, as shown in Section III-C.

B. Reference Architecture and Approach

In HLS DSE, we have to tackle a large number of benchmarks in order to explore the Pareto-optimal design. However, as the amount of data required for SCA grows exponentially along with the growth of design space, the traditional way for SCA security evaluation is extremely time-consuming and costly. Therefore, we propose a new methodology for HLS DSE. Our approach dramatically reduces the time and effort for SCA evaluation in memory-based HLS DSE by simplifying the SPN structure into a golden noise-free reference architecture, as shown in Fig. 5 (a). It has removed all external circuit that may cause undesirable noise in power side-channel while it keeps the basic features of an SPN structure.

The top-level block diagram of the reference architecture consists of four basic components: input/output signals, state registers, non-linear transformation module (Substitution Layer) and HLS/RTL benchmark IP library. The input signals include plaintext, encryption key, clock, and start signal. After being XORed, the intermediate values are loaded into the state registers. The output is the corresponding ciphertext taken from the state registers. For all benchmarks, we implement each design by replacing the corresponding non-linear submodule in the substitution layer. As the reference architecture uses state registers to store the intermediate values so the Hamming Distance between value updates in the registers can describe the real-world power consumption precisely, thus it is considered as a perfect power leakage model in this case.

C. Approach Generality and Experimental Limitation

The generality of the approach derives from the leakage mechanism of power side-channel. Considering the scenarios shown in Fig. 5 (b) and (c), some registers for loop pipelining

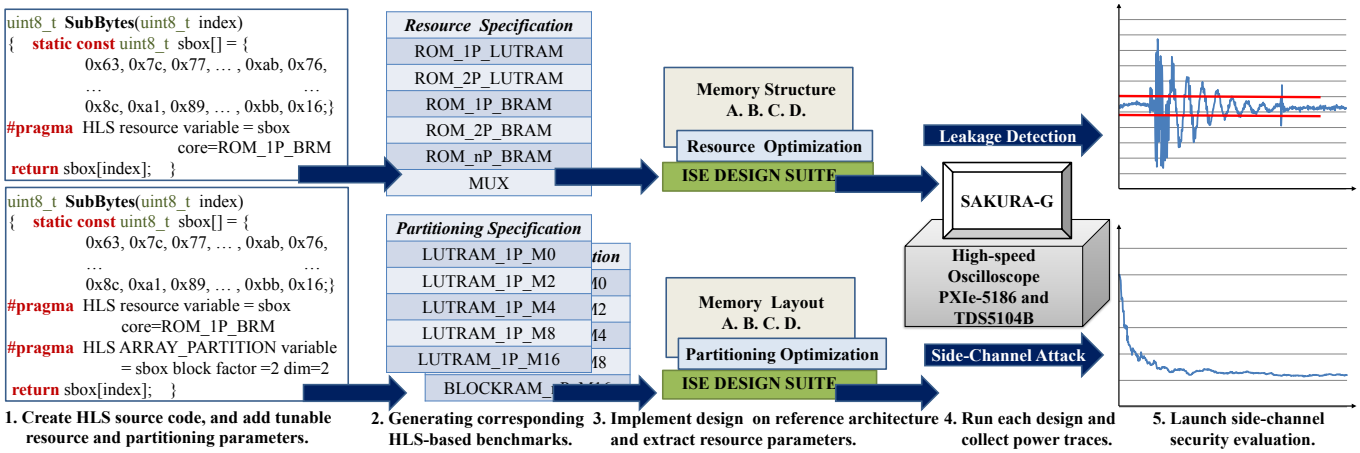
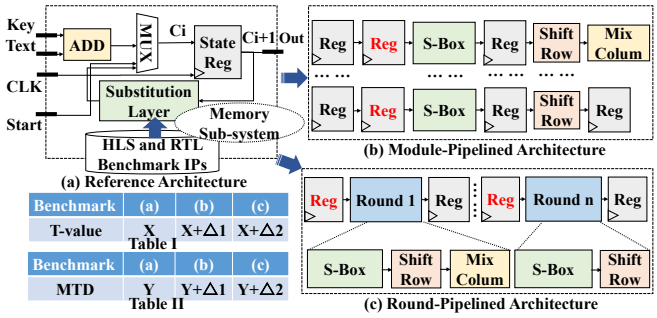


Fig. 4. The workflow to design each benchmark and perform the security evaluation. First, we create the benchmark using HLS or an existing RTL implementation. We evaluate only architectural changes related to the S-box, and therefore, we create a test harness focused on gathering the power consumption only in that module. The S-box architectures are synthesized to FPGA and executed with different power traces collected with an oscilloscope.



Benchmark	(a)	(b)	(c)
T-value	X	X+Δ1	X+Δ2

Table I

Benchmark	(a)	(b)	(c)
MTD	Y	Y+Δ1	Y+Δ2

Table II

Fig. 5. The implementation details of the proposed (a) Reference Architecture, (b) Module-Pipelined Architecture, and (c) Round-Pipelined Architecture.

and sub-modules are added to the original reference circuit in order to accelerate the throughput. In terms of the security, the side-channel vulnerabilities mainly originate from the value changes of the registers in red due to the fact that most power consumption is actually triggered by the registers at the internal side of substitution circuits. More importantly, the dynamic leakage consumption is actually generated due to the glitches happening in the following memory instances or combinational circuits that the internal registers supply. Therefore, the sub-architecture that includes the triggering registers and the substitution module within both pipelined architectures is equivalent to the proposed reference architecture from the perspective of side-channel leakage. Assume that, we can obtain the MTD and T-test results of architecture (a), (b), and (c), as shown in Table I and II of Fig. 5. Hence, by comparing the metric values mentioned above, Δ_1 and Δ_2 can precisely describe the side-channel effects due to the use of particular loop- or function-based HLS optimization.

In terms of the platforms, the primary combinational logic in both Altera and Xilinx FPGAs are equivalent, which is a dual-output 6-input LUT fabricated by TSMC. The only difference is that the 6-LUTs in Altera FPGA have more inputs than the ones in Xilinx FPGAs. For memory, some block RAMs in Xilinx are equivalent to the ones in Altera. For HLS tools, there might exist a little difference among the *performance-optimized* runs within various HLS tools, however, the side-channel leakage mechanism revealed by the exemplar of Vivado HLS is beneficial for understanding the side-channel effects in other scenarios. More importantly, the concept of constructing the golden noise-free design can be surely performed in other FPGA platforms as well as being beneficial for security evaluation in other HLS tools.

V. Investigated HLS Optimization Directives and Benchmarks Description

In this section, we briefly present the overview of memory space allocation and the notion of HLS DSE, then introduce how to perform local optimization using Vivado HLS as well as the details of each investigated HLS optimization directive.

A. Memory Space Allocation in HLS

HLS tools feature several optimizations to improve the design performance from the application perspective. Among them, the most important one that corresponds to side-channel security is the optimization with respect to memory space allocation. This type of HLS optimization hints the HLS tools to map or partition software data structures onto dedicated internal memories in order to achieve their design goals. By either changing the manual code *(e.g., BAMBU)* or configuring the local synthesis optimization, which is in the form of *pragmas (e.g., Vivado HLS and DWARV)* or *Tcl (e.g., LEGUP)* directives, designers can fully control the implementation of overall underlying micro-architecture at the behavioral level. Here, we consider the case of Vivado HLS as an exemplar to show the side-channel effects incurring from the use of different optimization strategies with respect to memory space allocation as well as generating the most optimal design.

B. HLS Resource Pragma Exploration

Pragma HLS resource in Vivado HLS enables us to determine which type of library resource is used to implement a variable or operation in RTL. Here, we use the memory

templates (hardware cores) expressed as RTL description in the component library of Vivado HLS to implement an array in C code. A template example would be

#pragma HLS resource variable=<name> core=<core> latency=<int>

where *variable* is an argument that specifies which object the pragma is assigned to, *core* is used to determine the hardware template in the library to implement the variable, and *latency* specifies the latency of the core. Then we can place the *pragma HLS resource* in the C source within the body of the function where the variable is defined. As a result, it allows us to control the transformation of a variable or operation in HLS-readable description. Therefore, by changing the options of the *core* argument, we could start our pragma-based DSE with respect to SCA security. Furthermore, the number of ports required for the memory expression in RTL can be determined by setting the HLS resource pragma as well. For example, if we want to implement an internal array (S-box) within "SubBytes" function of AES as a dual-port block RAM, we can configure the arguments as shown in Fig. 6 (left):

Original HLS-readable C codes

```
01   #include "AES.h"
02   // specify the resource from HLS
03   unit8_t SubBytes(unit8_t index)
04   {   static const unit8_t   sbox[ ] = {
05       0X63, 0X7c, … … , 0X76,
06         …                  …
07       0X8c, 0Xa1, … … , 0X16; }
08   #pragma HLS resource variable =
09     sbox  core = ROM_2P_BRAM
10     return sbox[index]; }
11   void main( ) {
12     SubBytes( ); ShiftRows( );
13     MixColums( );}
```

Dual-port block RAM RTL Description

```
01  (* rom_style = "block"*) reg [DW-1:0]
02                          ram[MEM_SIZE-1:0];
03  initial begin          //array initialization
04  $readmemh( "./SubBytes_rom.dat", ram);
05  end
06  always    @(posedge clk)
07    begin      //port-1 of block RAM
08      if (ce0)  begin  q0<=ram[addr0]; end
09    end
10  always    @(posedge clk)
11    begin     //port-2 of block RAM
12      if(ce1)   begin  q1<=ram[addr1]; end
13    end
```

Fig. 6. Code example for original HLS-readable C codes of "SubBytes" in AES (left) and HLS-generated dual-port block RAM RTL description (right).

Then we can obtain the RTL description of "SubBytes" function made by Vivado HLS, which describes the details of array initialization and array access for block RAMs, as shown in Fig. 6 (right). As an array within "SubBytes" function is only read as a static variable, it is typically implemented as a ROM after RTL synthesis. More precisely, during array initialization, the values within S-box are assigned to the block RAMs at the start of execution, which takes one clock cycle. Then each time the "SubBytes" function is executed, value is extracted from the corresponding location in the memory as the outputs, depending on the memory address provided as the inputs, which takes the other clock cycle. As we have set the block RAM as a dual-port instance, multiple elements can be accessed simultaneously in each clock cycle.

Consequently, by turning the argument *core* of *HLS resource pragma* to different template options, we can get a range of HLS-based architecturally unique benchmarks. In general, there exist three types of "S-box" implementations according to the hardware templates in Vivado HLS. 1) *ROM_LUTRAM:* one method is to implement a registered ROM of HLS using FPGA LUT memories, then achieving the functional goals by either loading the entries of the S-box to the ROM or fetching the desired value inside as S-box output. Each component in the registered ROM is accessed independent of the others, thus it can be configured either as one- or dual-port memory instance (e.g., *ROM_1P_LUTRAM*, *ROM_2P_LUTRAM*). Note that, multi-port architecture such as *ROM_nP_LUTRAM* can be generated, but practically, it can not work correctly due to the memory-port conflicts. Therefore, we do not take it into consideration. 2) *ROM_BRAM:* the other option is to use FPGA block RAM memories (i.e., *RAMB8BWER*); these memories are configurable as single- or dual-port instance (e.g., *ROM_1P_BRAM*, *ROM_2P_BRAM*), which follows the timing constraints of BRAM template in Vivado HLS. For *ROM_nP_BRAM* hardware template, adding ports increases the throughput while requiring more resources. 3) *MUX:* Another technique is to store the S-box entries into the FPGA fabric as constants and use a multiplexer to decide between them. This architecture is implemented as non-registered combinational logic, thus only one clock cycle is used to complete the entire function.

To demonstrate a parallel controlled trial, we further perform experiments on five hand-written RTL-based architectures, which are from currently available open-source benchmarks for side-channel analysis [21]. All those RTL-based benchmarks are implemented using LUT primitives (e.g., LUT-4, LUT-6, etc. for Xilinx FPGA) and not directly driven by any clock signal. For details, LUT benchmark describes the S-box by using case statement sentence. And COMP benchmark is implemented by using composite field implementation based on multiplicative inverse circuit. While PPRM1 and PPRM3 are both performed using Positive Polarity Reed-Muler logic, but with one and three logic stages, respectively. WDDL is achieved using wave dynamic differential logic, as a famous countermeasure against first-order SCA attacks [22].

C. HLS Partition Pragma Exploration

Pragma HLS array_partition in Vivado HLS provides commendations on how to split an entire array into smaller arrays or individual elements. Here, we use the *array_partition* pragma to partition the array in C code into a variety of multi-memory architectures. A template example would be

#pragma HLS array_partition variable=<name> <type> factor=<int> dim=<int>

where *variable* is an argument that determines the array to be partitioned, *type* is used to specify the partition strategy provided by Vivado HLS, *factor* controls the number of the smaller arrays to be created, and *dim* allows a multi-dimensional array to determine which dimension is to be partitioned. Then it is used by placing the *array_partition* pragma in the boundaries of function where the array is defined. Consequently, changing both the *type* argument and the *factor* argument of the pragma *array_partition* allows us to explore the security aspects of the memory sub-system.

In RTL-level description, applying the *array_partition* directive results in architecture with multiple small memories or registers instead of one large memory. Vivado HLS supports three styles of partitioning schemes expressed as *block*, *cyclic* and *complete*. For example, consider a simple case of a 2-dimensional array with 6×6 elements. In *block* scheme, contiguous elements in the original array are divided equally.

In *cyclic* scheme, interleaving blocks of the original array are selected to constitute the partitioned memory banks. In *complete* scheme, each element in the original array is separated individually. As a consequence, it could effectively increase the amount of the read- and write-port of the storage, thus potentially improving the throughput of a design. However, the number of memory instances alongside the scale of memory control logic increases simultaneously as the design penalties.

To figure out more insights about how array partitioning affects the power side-channel leakage, we combine the pragma *resource* and pragma *array_partition* together as optimization strategies for the original design to obtain a range of multi-memory architectures. Although each multi-memory architecture is different depending on its specific optimization configuration, the top-level block diagram of a partitioned memory system basically shares the same structure. It is generally composed of memory banks, address translation unit, control *FSM*, read/write registers and input/output *MUXs*. For example, assuming to partition an S-box, which holds 256 elements inside, into four parallel memory banks by setting the *factor* argument to 4 and *dim* argument to 2. Thus, each memory bank stores 64 elements, and all the banks are accessed simultaneously. Following those memory banks is the memory control logic, which is instantiated as the combinations of *MUX* and *LUT* primitives. Note that, there exists no obviously architectural difference between *block partition* and *cyclic partition* in terms of SCA security. Therefore, we only take one of them into consideration in the SCA evaluation.

TABLE I
PRAGMA-BASED CONFIGURATION AND BENCHMARKS DESCRIPTION.

Benchmarks(HLS)	Resource Template	Partition Scheme & Num.
LUTRAM1P_M0	ROM_1P_LUTRAM	Complete/Unpartitioned/0
LUTRAM1P_M2		Block/Cyclic/2
LUTRAM1P_M4		Block/Cyclic/4
LUTRAM1P_M8		Block/Cyclic/8
LUTRAM1P_M16		Block/Cyclic/16

With respect to the number of partitioned banks, massive architecturally unique designs can be created by specifying the corresponding partition scheme and partition number, respectively. Here, we choose the "ROM_1P_LUTRAM" HLS template as an exemplar. As shown in Table I, we can see that four benchmarks (from LUTRAM1P_M2 to _M16)) with different partition configuration are derived from the same baseline architecture (LUTRAM1P_M0), which is generated by applying a specific *pragma resource* template.

VI. PRACTICAL EVALUATION

In this section, we provide an evaluation of different benchmarks using a testing framework that includes the same test harness, *SAKURA* board, oscilloscope, evaluation and attack (see Section IV-A). We perform pragma-based HLS DSE among those architectures with respect to throughput, resource utilization, and power side-channel resilience in Section VI-B. To determine the security of the design, we evaluate each benchmark using first-order leakage detection in Section VI-C and first-order CPA attack in Section VI-D.

A. Experimental Setup

We implemented all the benchmarks depicted above on the widely-used side-channel evaluation board *SAKURA-G*, which featuring a Xilinx Spartan-6 *XC6SLX75* FPGA for cryptography implementation and a Spartan-6 *XC6SLX9* FPGA as the controller, as shown in Fig. 7. The implementation in Fig. 5 (a) is considered as the reference architecture. Moreover, all the implementations were served using 24MHz clock frequency. In order to sample aligned power traces, reference design will provide a control signal for measurement triggering. The evaluation board is connected with Host PC through the USB interface for data communication. Then the leakage traces are recorded by means of TDS5104B and PXIe-5186 as high-speed oscilloscopes at a sampling rate of 1GS/s and a differential probe is used to monitor the voltage drop by a 1 Ω resistor from measurement point J3 on the board.

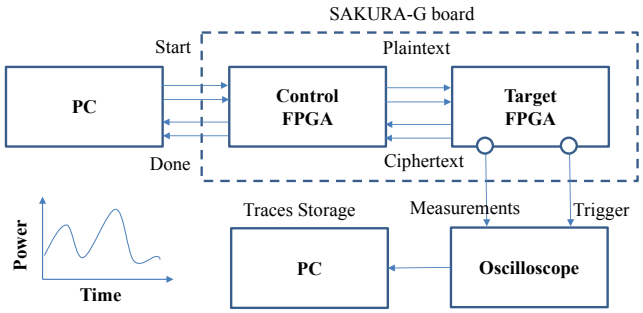


Fig. 7. Experimental setup demonstrating the SAKURA-G board, our measurement oscilloscope and PC for data acquisition.

B. Design Space Analysis

We performed a pragma-based security DSE of HLS optimizations by considering three metrics: *Throughput*, *Resource Usage* and *Security Metrics (Leakage & MTD)*. To compare the benchmarks for HLS DSE, we measure the benchmarks by following the criteria below.

1) Security dominance (SD): security is highlighted as a dominating aspect in HLS DSE by considering both qualitative (e.g., a confidence level) and quantitative (e.g., MTD) security metrics. This means that if a design is both qualitative and quantitative secure, then it is considered as the dominating design. If a design only satisfies one of the security metrics, it is considered to be the less optimal one. If neither of the security metrics is satisfied, it is considered as an insecure one even if other performances are considerably good enough.

2) Pareto Dominance (PD): assume the scenario that the benchmarks have either qualitative or quantitative SCA leakage, then we would follow the PD criteria to rank these architectures. To give a quantitative criterion, we measure the PD dominating design by calculating the equation (3):

$$PD(T, R, M) = \frac{T}{T_m} + \frac{R_m - R}{R_m} + \frac{M}{M_m} \tag{3}$$

where T denotes the throughput, T_m denotes the maximum of throughput, which is a constant with the value 128; R denotes the sum of resource usage, R_m denotes the maximum

sum of resource usage, which is a constant with the value 998; M denotes the number of measurements to disclosure the secrets, M_m denotes the maximum of MTD, which is a constant with the value 1339. The higher the PD value is, the more Pareto-optimal the design is.

Table II describes the *pragma HLS resource*-oriented benchmarks. Our goal is to evaluate the spectrum of architectural choices for the S-boxes. These include benchmarks from existing RTL implementations (the first five labeled as "RTL") and HLS-based benchmarks (the last six labeled as "HLS"). The first column of Table II shows the name of the architecture, the second column gives the throughput, column three reports the resource usage (Slices/LUTs/BRAM), and column four shows metrics related to its vulnerability to power SCAs – "Leakage" is related to the t-test and denotes whether the design shows a difference in the t-test results; a secure design has NO leakage. MTD denotes the number of traces required to recover the key.

TABLE II
THROUGHPUT, RESOURCE USAGE, AND POWER SIDE-CHANNEL LEAKAGE OF DIFFERENT S-BOX ARCHITECTURES. WE EVALUATE TWO METRICS FOR SECURITY – USING THE T-TEST (YES MEANS LIKELY LEAKAGE) AND THE NUMBER OF MEASUREMENTS TO DISCLOSURE (MTD) USING CPA.

Benchmarks	Throughput	Resource	Leakage & MTD
LUT(RTL)	8bits/cycle	8/32/0	YES/133
COMP(RTL)	8bits/cycle	23/54/0	YES/293
PPRM1(RTL)	8bits/cycle	99/216/0	YES/55
PPRM3(RTL)	8bits/cycle	26/53/0	YES/163
WDDL(RTL)	8bits/cycle	150/432/0	NO/100,000+
MUX(HLS)	8bits/cycle	9/9/0	YES/340
LUTRAM1P(HLS)	8bits/cycle	12/32/0	YES/165
LUTRAM2P(HLS)	16bits/cycle	88/179/0	YES/294
BlockRAM1P(HLS)	8bits/cycle	0/0/1	YES/100,000+
BlockRAM2P(HLS)	16bits/cycle	0/0/1	YES/100,000+
BlockRAMnP(HLS)	128bits/cycle	0/0/8	NO/100,000+

Remarks: *This experiment shows that the implementation of S-boxes not only affects the throughput and the sum of resource usage but also incurring different levels of security risks. Note that, most benchmarks, which are primarily implemented using the* LUTs*, all could be hacked successfully, regardless of whether the benchmarks are RTL hand-written or HLS-generated. That means the* LUTs*, which are well-adopted logic primitives, show a natural weakness for attackers. However, the SCA threats also vary greatly depending on the implementation method and the number of resources. For instance, the WDDL benchmark, mostly implemented by* LUTs*, also shows robustness against SCAs due to the use of dynamic differential logic, which can effectively balance the switching events of logic gates. For benchmarks using block RAMs, they all show their natural robustness against SCAs. Among them, the* BlockRAMnP *in Vivado HLS is the Pareto-optimal design as it consumes a small number of resources but has the largest throughput. It is the most secure design as it neither shows first-order leakage nor leads to any key recoveries.*

Table III summarizes the benchmarks obtained by applying different combinations of *pragma HLS resource* and *pragma HLS array_partition* strategies, where the first column describes the name of the benchmarks, the second column gives the number of partitioned banks, the third column shows the partition scheme applied on each benchmark, the fourth column presents the throughput, the eighth column reports the *"Leakage"* and *"MTD"* metrics, the ninth column shows the *PD value* or *SD label*, and the remaining columns indicate the number of resource usage - *Slice Reg*, *LUTs*, *BRAM*, respectively. For the ease of comparison, we visualized these results according to three measures: *MTD*, *Throughput*, and *Resource Usage* in Fig. 8. The upper row demonstrates the *MTD* comparison for each *"resource"* series of benchmarks. The lower row illustrates the *Throughput* and *Resource Usage* comparisons among all the series. Fig. 9 demonstrates the comparison of PD values for all the benchmarks.

1) ***HLS Partition and SCA Security:*** *Note that, in the upper row of Fig. 8, we can see that, regardless of which* "resource" *series (*LUTRAM *or* BlockRAM*) they belong to, all the benchmarks become SCA-attackable after partitioning into multi-memory architecture using* HLS array_partition *optimization. It indicates that the partition optimization in Vivado HLS could compromise your hardware system in terms of SCA security, regardless of which types of resource primitives are used to achieve the designs. Moreover, with the increase of the partition number, the MTD value decreases sharply due to the multi-memory layout. It is particularly obvious for the* BlockRAM-series *benchmarks because the benchmarks all show strong robustness before partitioning. However, over 60% of the benchmarks are SCA-attackable after partitioning.*

2) ***Leakage Source and Noise Analysis:*** *To figure out the source of the SCA leakage, we further make comparisons between* Resource Usage *and* MTD*. More precisely, in Fig. 8, we can see that, with the growth of* LUTs*, the* MTD *values decrease obviously, indicating that the number of* LUTs *plays an important role in the robustness against SCA attacks. For instance, considering the case of* BlockRAM1P *and* BlockRAM2P *series,* BlockRAM1P_M0 *and* BlockRAM2P_M0 *have the same number of* "Slice Reg" *and* "BRAM"*, however, since dual-port block RAM requires more* LUTs *to establish the memory control logic, the benchmarks in* BlockRAM2P *series show 42% reduction in average* MTD *value (more vulnerable) compared with the corresponding MTD values in* BlockRAM1P *series. We can thus assume that the side-channel leakage is essentially from the deployment of memory controller logic rather than saying it is from the block RAMs themselves. Indeed, the number of* block RAMs *obviously shows more effects on the environmental noise level.*

3) ***Security Dominance and Pareto Dominance:*** *In Fig. 9, it shows that, among all the PD-marked benchmarks, the Pareto dominance is* BlockRAM1P_M4*, which has the highest PD value of 1.823. Meanwhile, we can obtain five* Security Dominance *designs, which are* BlockRAM1P_M0*,* BlockRAM1P_M2*,* BlockRAM2P_M0*,* BlockRAMnP_M0*,* BlockRAMnP_M2*, respectively. To evaluate the security level of these architectures, we further delve deeper into the leakage detectability analysis in the following section.*

C. Leakage Detectability Analysis

We performed the non-specific t-test as described in Section III-B. One straight benefit of non-specific t-test is its natural

TABLE III

THROUGHPUT, RESOURCE USAGE, AND POWER SIDE-CHANNEL LEAKAGE OF DIFFERENT S-BOX ARCHITECTURES. WE EVALUATE TWO METRICS FOR SECURITY – USING THE T-TEST (YES MEANS LIKELY LEAKAGE) AND THE NUMBER OF MEASUREMENTS TO DISCLOSURE (MTD) USING CPA.

Benchmarks(HLS)	Partition Num.	Partition Scheme	Throughput	Slice Reg	LUTs	BRAM	Leakage & MTD	PD
LUTRAM1P_(M0-M16)	0	Complete/Unpartitioned	8bits/cycle	146	174	0	YES/165	0.865
	2	Block/Cyclic	8bits/cycle	155	183	0	YES/136	0.825
	4	Block/Cyclic	8bits/cycle	170	197	0	YES/249	0.881
	8	Block/Cyclic	8bits/cycle	202	233	0	YES/269	0.828
	16	Block/Cyclic	8bits/cycle	266	315	0	YES/219	0.644
LUTRAM2P_(M0-M16)	0	Complete/Unpartitioned	16bits/cycle	154	263	0	YES/294	0.927
	2	Block/Cyclic	16bits/cycle	172	224	0	YES/1008	1.481
	4	Block/Cyclic	16bits/cycle	202	251	0	YES/573	1.099
	8	Block/Cyclic	16bits/cycle	266	324	0	YES/450	0.870
	16	Block/Cyclic	16bits/cycle	395	488	0	YES/124	0.333
BlockRAM1P_(M0-M16)	0	Complete/Unpartitioned	8bits/cycle	138	80	1	YES/100,000+	SD
	2	Block/Cyclic	8bits/cycle	139	81	2	YES/100,000+	SD
	4	Block/Cyclic	8bits/cycle	138	97	4	YES/1339	1.823
	8	Block/Cyclic	8bits/cycle	138	99	8	YES/512	1.199
	16	Block/Cyclic	8bits/cycle	138	117	16	YES/392	1.084
BlockRAM2P_(M0-M16)	0	Complete/Unpartitioned	16bits/cycle	138	84	1	YES/100,000+	SD
	2	Block/Cyclic	16bits/cycle	140	86	2	YES/1011	1.652
	4	Block/Cyclic	16bits/cycle	138	118	4	YES/760	1.432
	8	Block/Cyclic	16bits/cycle	138	122	8	YES/407	1.160
	16	Block/Cyclic	16bits/cycle	138	158	16	YES/145	0.921
BlockRAMnP_(M0-M16)	0	Complete/Unpartitioned	128bits/cycle	138	140	8	NO/100,000+	SD
	2	Block/Cyclic	128bits/cycle	154	223	16	YES/100,000+	SD
	4	Block/Cyclic	128bits/cycle	138	412	32	YES/238	1.595
	8	Block/Cyclic	128bits/cycle	138	444	64	YES/192	1.496
	16	Block/Cyclic	128bits/cycle	138	732	128	YES/281	1.210

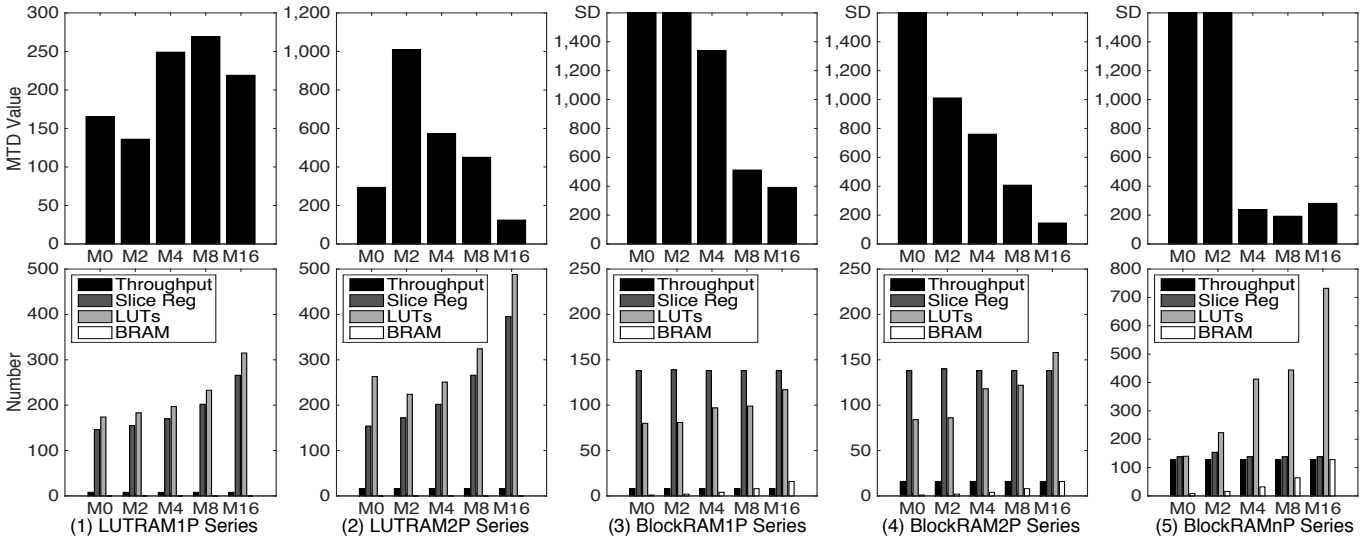


Fig. 8. The comparisons of SCA security evaluation metric (MTD), Throughput, Slice Reg, LUTs, BRAM among all HLS-generated benchmark series.

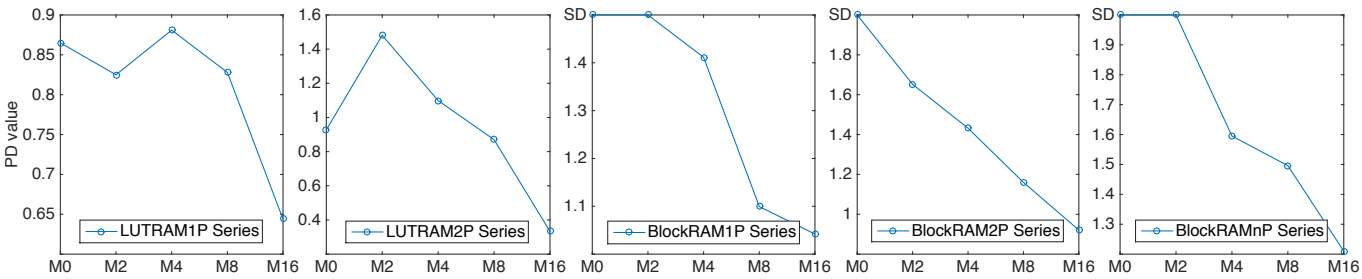


Fig. 9. The comparison of PD dominance design and SD dominance design among all HLS-generated benchmark series.

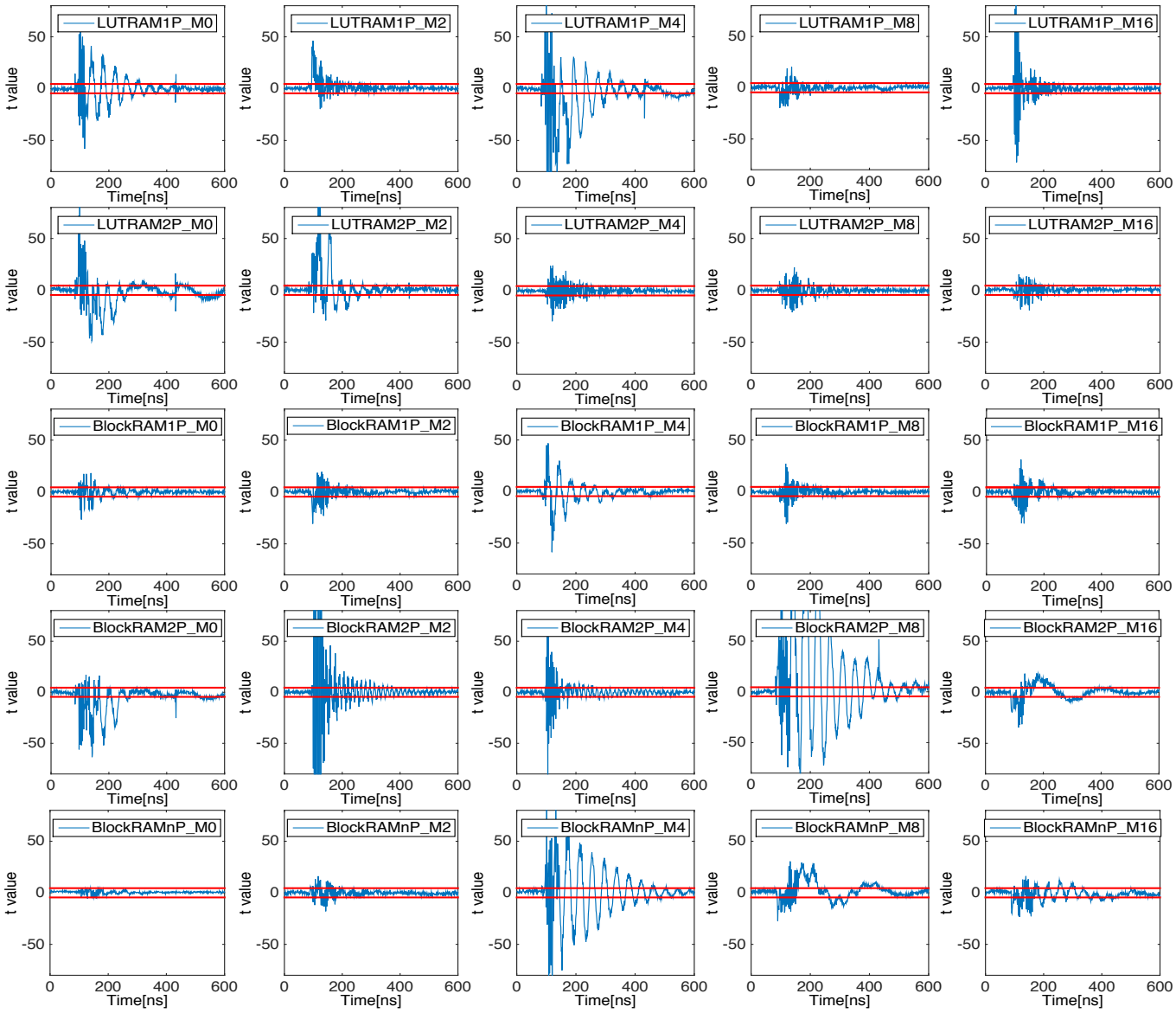


Fig. 10. First-order side-channel leakage detection results. The y-axis denotes the t-test result for that specific time. A large t value indicates a higher chance for leakage detection. The red threshold lines show the $|t| > 4.5$, which is the widely used value to indicate that the t-test rejects the null hypothesis.

independence from any estimated power models, which allows for more fair and efficient comparisons for HLS DSE. Practically, it could identify stable first-order leakage after using 100,000 traces. Fig. 10 depicts the details of first-order leakage among all HLS-generated benchmarks.

To rank the security level of these *Security Dominance* architectures, we compare these *SD benchmarks* by following the criterion below. For instance, as seen in Fig. 10, *BlockRAM2P_M0* is the least optimal one as it has a higher leakage peak and a wider time range of leakage. While *BlockRAMnP_M0* is the most optimal design as the *"leakage"* is only floating inside the leakage boundaries of t-test (highlighted in red), indicating that there exists no first-order leakage. Following the same criterion, the rank is *BlockRAMnP_M0*, *BlockRAMnP_M2*, *BlockRAM1P_M0*, *BlockRAM1P_M2*, *BlockRAM2P_M0*, respectively (from most optimal to least optimal). In addition, we can observe that the benchmarks in *BlockRAM2P series* obviously show more informative leakage than the benchmarks in other *BlockRAM-based* series. This observation reconfirms our assumption that, with the growing number of partitions, the scales of memory control logic increase sharply due to the use of dual-port memory instance, which simultaneously leading to more *"LUTs"* consumption, finally resulting in a more vulnerable architecture for SCA attackers. Note that, when the number of partitions is set as 2, the benchmark in *BlockRAMnP series* show less first-order leakage since the noises originating from the block RAMs have a predominant effect on the power side-channel in this case. However, with the growth in the number of partitions, the leakage effects caused by the controller logic become the predominant one, which results in more obvious leakage for other benchmarks in *BlockRAM_nP series*.

Remarks: *These experiments show that the use of dual-port block RAM can incur larger first-order side-channel leakage.*

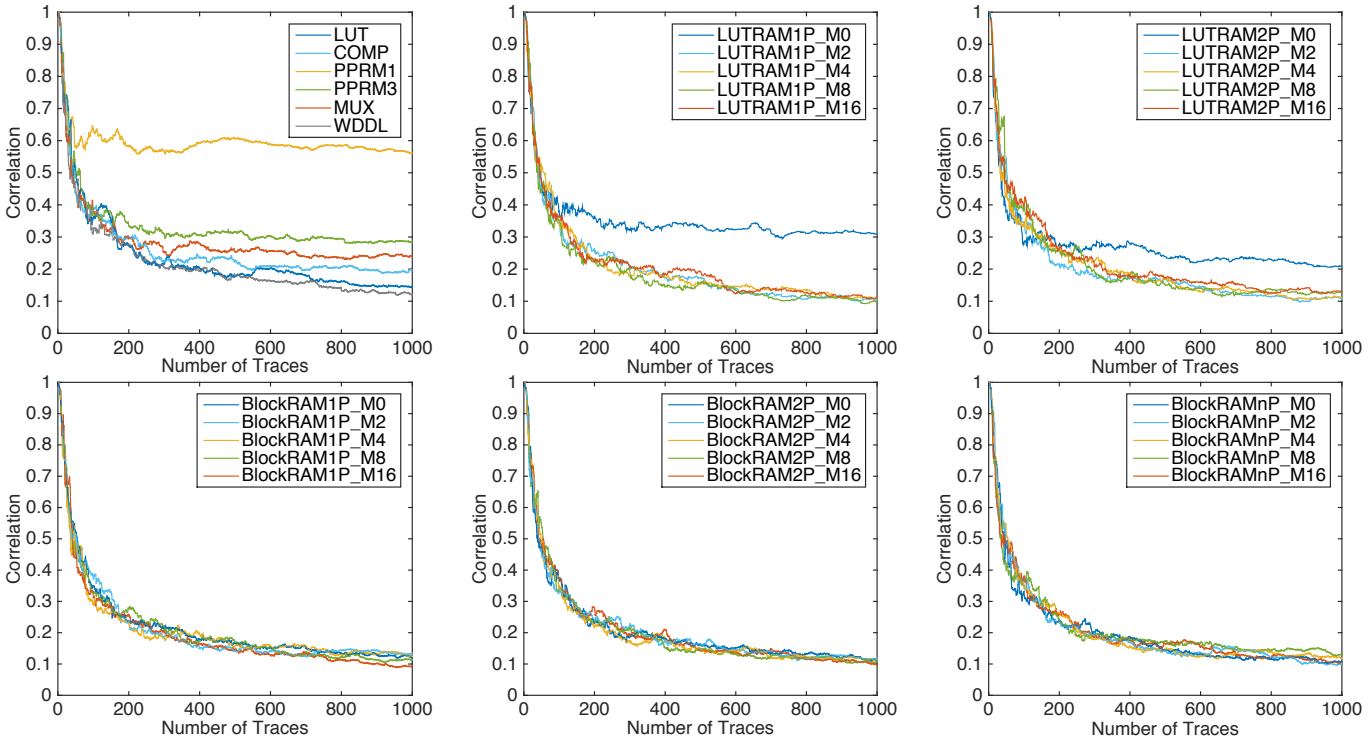


Fig. 11. Single key byte result of a first-order CPA on benchmarks with different combination of optimization strategies and the hand-written RTL benchmarks.

Thus it is more efficient and secure implementing the design with 8 paralleled single-port block RAMs due to the fact that the noise level increases significantly so that the leakage effects are diminished or counteracted in this case. For BlockRAM1P *and* BlockRAMnP *templates in Vivado HLS, setting the number of partitions as 0 and 2 produces good results in terms of side-channel security. However, applying* array_partition *in other cases could incur serious side-channel challenges. Therefore, one security guideline for HLS designers is to avoid partitioning the arrays or to be more careful about accessing 2 more memories that supply a combinational logic. It is both true for the LUTRAM and BlockRAM instances in Vivado HLS.*

D. Quantitative Leakage Analysis

We performed the first-order non-profiled CPA attack, as described in Section III-C, on these benchmarks with known randomly generated plaintexts and a fixed key. The practical attack was presented following a divide-and-conquer approach where each key byte is attacked in isolation. Our attack models the dynamic consumption as $HD(c_{i+1} \bigoplus c_i)$, where HD denotes the Hamming Distance, c_i and c_{i+1} are the intermediate input and output, respectively. Then we attempt to identify the correct key guess on a byte-by-byte basis by applying Pearson's correlation as distinguisher and simultaneously record the number of traces required to recover the key in each architecturally unique design as the security metric (MTD) for further security-based HLS DSE.

Fig. 11 illustrates the detailed correlation values and the general trend of the correlation value for all the benchmarks across a single key byte. We can observe the effects of *HLS pragma resource* and *HLS pragma array_partition* on the correlation coefficient values among all benchmarks. The MTD results of CPA attack among all these benchmarks are listed in detail, as shown in Table III. We discuss it together with HLS DSE in Section VI-B. Here, we mainly focus on the effect of applying *HLS pragma resource* and *HLS pragma array_partition* on the Pearson's correlation coefficient.

From results shown in the upper left of Fig. 11, one important observation is that all the curves of correlation finally converge to the different end value, indicating that the uniqueness of architecture can lead to different levels of SCA leakage and environmental noise. The upper middle and upper right of Fig. 11 demonstrate the correlation details of benchmarks in LUTRAM1P series and LUTRAM2P series, respectively. We can see that the curves of Pearson's correlation coefficient for the partitioned benchmarks all converge to the lower end values in comparison with the unpartitioned M0 benchmark, indicating that the noise level of the designs increases obviously with the growth of memory partitions. The lower three illustrations of Fig. 11 demonstrate that all the *BlockRAM-based* benchmarks have the similar curve feature. Although the block RAM primitives show their natural robustness against SCA attacks, the connection between partitioned block RAMs and controller logic results in unpredictable security risks.

Remarks: *The experiment shows that array partition in HLS tools can significantly affect the side-channel vulnerabilities. For HLS designers, since memory-based optimization in HLS can obviously change the memory sub-system, it is better to consider the security of the entire sub-system including the controller logic and memory banks rather than only focusing on one of them, respectively. Apart from performance metrics, security metrics should also be highlighted in order to create*

fast, small, and secure cryptographic design with HLS tools. Hence, HLS designers can quantitatively balance the security in exchange of performance depending on the application scenarios. For side-channel attackers, it is more beneficial to attack the partitioned architectures generated by HLS tools. Because up to 90% of the partitioned architectures might exist unknown backdoors that potentially to be the leakage spots.

VII. Related works

There are numerous efforts focusing on providing best-effort optimization strategies to generate high-quality architectures using HLS techniques. Li et al. [23] and Cong et al. [24] performed a comprehensive study on behavioral synthesis optimizations. They demonstrated a better combination of various HLS optimization strategies to obtain good performance. Wang et al. [25] used an automatic memory partitioning scheme to achieve an optimal design with high data throughput and a small logic overhead. Cilardo et al. [26] better leveraged the partitioning and unrolling optimizations to reduce the area overhead of a specific design. However, most of them focus on the performance enhancement of throughput and area, and none of them consider the problem in terms of SCA security.

There also exist various works that spare pretty much efforts on providing efficient side-channel leakage detection methodologies or practical side-channel attacks. Goodwill et al. [18] used the statistical hypothesis testing as a standardized testing program to detect potential side-channel vulnerabilities in your design. Doget et al. [20] provided valuable comparisons in terms of attack efficiency among most univariate attacks in the literature. Standaert et al. [27] addressed the serious side-channel threats for FPGA-security using well-known side-channel attacks. Rostami et al. [28] summarized the models, methods and evaluation metrics for most hardware-based attacks. Tang et al. [29] proposed some security metrics to evaluate the tamper resistance of the pin mapping algorithms. Yet, none of those works attempt to characterize the side-channel effects caused by behavioral synthesis as well as the relevant security metrics, aiming to evaluate the consequences of applying those memory-based HLS optimization.

For HLS DSE and memory-based optimization, Pilato et al. [30] provided a system-level optimization for a memory system in order to automatically generate more efficient architectures by means of their proposed methodology for HLS DSE. Schafer [12] performed a new method to accelerate the HLS DSE by classifying the HLS optimization knobs, and he also performed an HLS resource sharing DSE by fixing the bitwidth of internal variables in [31]. However, none of those work focus on the SCA security DSE problem in HLS, there thus exists a lack of guidelines for security-based HLS DSE.

Perhaps the most relevant work to ours is that done by Sun et al. [32] and Homsirikamol et al. [4]. They are trying to determine the most suitable HLS approach to implement high-quality hardware cryptographic cores with minimal development effort. This is similar in spirit to the goal in our work since we also aim to provide comprehensive guidance to optimize the HLS workflow. However, we focus more on evaluating the consequences of those HLS optimizations from the perspective of security, while other researchers show more interests on the performance increment under an area budget.

VIII. Conclusion

In this paper, we investigated the effects of memory-based architectural optimization on power side-channel leakage. We developed a workflow to properly gather power traces. We generated a set of representative benchmarks that employ different S-box architectural optimizations. We provide a comparison between these different architectures in terms of "traditional" design metrics of performance and resource usage alongside the security metric related to power side-channel leakage. This enables us to explore the design space and provide concrete recommendations on architectures that are efficient with respect to performance, resource usage, and security. Future work will delve more architectural optimizations.

References

[1] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 18–25, 2009.

[2] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on fpgas," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, 2016, pp. 918–923.

[3] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen, "Design and implementation of low-area and low-power aes encryption hardware core," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006, pp. 577–583.

[4] E. Homsirikamol and K. Gaj, "Can high-level synthesis compete against a hand-written code in the cryptographic domain? a case study," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, 2014, pp. 1–8.

[5] L. Piccolboni, G. Di Guglielmo, and L. P. Carloni, "Pagurus: Low-overhead dynamic information flow tracking on loosely coupled accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2685–2696, 2018.

[6] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2007, vol. 31.

[7] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-analysis attack on an asic aes implementation," in *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, vol. 2, 2004, pp. 546–552.

[8] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked aes hardware implementations," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2005, pp. 157–171.

[9] W. Puech, M. Chaumont, and O. Strauss, "A reversible data hiding method for encrypted images," in *Electronic Imaging*, no. 6819, 2008, p. 68191E.

[10] V.-H. Xilinx, "Vivado design suite user guide-high-level synthesis," 2014.

[11] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "Legup: high-level synthesis for fpga-based processor/accelerator systems," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2011, pp. 33–36.

[12] B. C. Schafer, "Probabilistic multiknob high-level synthesis design space exploration acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 394–406, 2016.

[13] L. Zhang, W. Hu, A. Ardeshiricham, Y. Tai, J. Blackstone, D. Mu, and R. Kastner, "Examining the consequences of high-level synthesis optimizations on power side-channel," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, March 2018, pp. 1167–1170.

[14] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.

[15] R. Nane, V.-M. Sima, B. Olivier, R. Meeuws, Y. Yankova, and K. Bertels, "Dwarv 2.0: A cosy-based c-to-vhdl hardware compiler," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2012, pp. 619–622.

[16] C. Pilato and F. Ferrandi, "Bambu: A modular framework for the high level synthesis of memory-intensive applications," in *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–4.

[17] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi *et al.*, "Test vector leakage assessment (tvla) methodology in practice," in *International Cryptographic Module Conference*, vol. 1001, 2013, p. 13.

[18] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, 2011, pp. 158–172.

[19] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[20] J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert, "Univariate side channel attacks and leakage modeling," in *Journal of Cryptographic Engineering*, vol. 1, no. 2, 2011, pp. 123–144.

[21] S. Morioka and A. Satoh, "An optimized s-box circuit architecture for low power aes design," in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2002, pp. 172–186.

[22] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure dpa resistant asic or fpga implementation," in *Design, Automation & Test in Europe Conference & Exhibition*, 2004.

[23] P. Li, L.-N. Pouchet, and J. Cong, "Throughput optimization for high-level synthesis using resource constraints," in *Int. Workshop on Polyhedral Compilation Techniques (IMPACT'14)*, 2014.

[24] J. Cong, M. Huang, P. Pan, Y. Wang, and P. Zhang, "Source-to-source optimization for hls," in *FPGAs for Software Programmers*. Springer, 2016, pp. 137–163.

[25] Y. Wang, P. Li, P. Zhang, C. Zhang, and J. Cong, "Memory partitioning for multidimensional arrays in high-level synthesis," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 12.

[26] A. Cilardo and L. Gallo, "Interplay of loop unrolling and multidimensional memory partitioning in hls," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 163–168.

[27] F.-X. Standaert, L. v. O. tot Oldenzeel, D. Samyde, and J.-J. Quisquater, "Power analysis of fpgas: How practical is the attack?" in *International Conference on Field Programmable Logic and Applications*. Springer, 2003, pp. 701–710.

[28] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2015.

[29] J. Tang, M. Ibrahim, K. Chakrabarty, and R. Karri, "Synthesis of tamper-resistant pin-constrained digital microfluidic biochips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[30] C. Pilato, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "System-level optimization of accelerator local memory for heterogeneous systems-on-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 435–448, 2017.

[31] B. C. Schafer, "Enabling high-level synthesis resource sharing design space exploration in fpgas through automatic internal bitwidth adjustments," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 97–105, 2017.

[32] Z. Sun, K. Campbell, W. Zuo, K. Rupnow, S. Gurumani, F. Doucet, and D. Chen, "Designing high-quality hardware on a development effort budget: A study of the current state of high-level synthesis," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016, pp. 218–225.

PLACE PHOTO HERE

Lu Zhang received his B.S. degree from Northwestern Polytechnical University, Xi'an, Shaanxi, China, in 2012, where he is currently pursuing his Ph.D. degree. From January 2016 to January 2018, he was a visiting graduate student with the Department of Computer Science and Engineering, University of California, San Diego.

His research interests include hardware security, design automation and embedded systems and optimization.

PLACE PHOTO HERE

Dejun Mu received the Ph.D. degree in Control Theory and Control Engineering from Northwestern Polytechnical University, Xi'an, Shaanxi, China, in 1994. He is currently a Professor with the School of Cyberspace, Northwestern Polytechnical University, China.

His current research interests include control theories and information security, including network information security, application specific chips for information security, and network control systems.

PLACE PHOTO HERE

Wei Hu received his Ph.D. degree in Control Science and Engineering from the Northwestern Polytechnical University, Xi'an, Shaanxi, China, in 2012. He is currently an associate professor with the School of Cyberspace, Northwestern Polytechnical University, China.

His research interests include hardware security, logic synthesis, and formal verification, reconfigurable computing and embedded systems.

PLACE PHOTO HERE

Yu Tai received his Ph.D. degree in Control Science and Engineering from Northwestern Polytechnical University, Xi'an, Shaanxi, China, in 2018. He is currently a Research Assistant with the School of Cyberspace, Northwestern Polytechnical University, China.

His current research interests include hardware security, logic synthesis and optimization in hardware information flow.

PLACE PHOTO HERE

Jeremy Blackstone is a Ph.D. student from the Computer Science and Engineering, University of California, San Diego. He received his B.S. and Master degree in computer science from Howard University in Washington, DC.

His current research interests include hardware security and fault attacks.

PLACE PHOTO HERE

Ryan Kastner received the Ph.D. degree in Computer Science from the University of California at Los Angeles, Los Angeles, CA, USA, in 2002. He is currently a Professor with the Department of Computer Science and Engineering, University of California at San Diego, San Diego, CA, USA. Prof. Kastner is the Co-Director of the Wireless Embedded Systems Master of Advanced Studies Program. He also codirects the Engineers for Exploration Program.

His current research interests include hardware acceleration, hardware security, and remote sensing.